Apple IIGS
#18: Do-It-Yourself SCC Access

Revised by: Jim Luther July 1990
Written by: Jim Luther, Mike Askins, Matt Deatherage & Jim Mensch June 1987

This Technical Note describes how to install and remove a interrupt handler routine for the Z8530 Serial Communications Controller (SCC) on the Apple IIGS without breaking other parts of the system. This Note includes many suggestions that, if unheeded, could come back to haunt you in the form of bug fixes to your program.

Changes since March 1990: Added a method for finding which serial port AppleTalk is using under GS/OS.

Free Serial Routines Inside

The Z8530 SCC has 2 serial channels, supports several synchronous and asynchronous data communications protocols, and has 9 read registers and 16 write registers per channel. (Compare this to the 5 registers of the 6551 Asynchronous Communications Interface Adapter.) To program the SCC correctly, you must understand five things: the SCC, the Apple IIGS hardware environment in which the SCC lives, the Apple IIGS interrupt handler firmware, the interrupt support provided by the operating system, and the data communication protocol you want to use. If you don't understand all of these components, stick to the serial firmware.

The Apple IIGS serial firmware is a robust environment for almost every asynchronous serial programming application. If you want to handle all SCC operations and SCC interrupts on the IIGS without using the serial firmware, then you must really **know** the firmware won't do the job for you or you wouldn't be going to a lot of trouble to recreate the services the firmware routines already provide.

Don't Eat Your Serial with Your Mouth Open

Your mother has rules and so does Apple. On many systems, your application may be sharing the SCC chip with System Software such as AppleTalk or the serial firmware. If you want to access the SCC chip directly without breaking the system (or the system breaking you), then follow these simple rules.

Rule #1: Before using a serial port, make sure AppleTalk is not already using it.

If AppleTalk is active, it uses one of the serial ports. The user selects which serial port AppleTalk uses with the Control Panel. Before using one of the serial ports, you should always check to make sure AppleTalk is not using that port. If AppleTalk is using the serial port your application wants to use, tough luck; tell the user about it, but don't even think about using that port.

Under ProDOS 8, use the method shown in the following sample code to determine if AppleTalk is using a serial port:

```
;
; This routine checks to see which serial port, if any, AppleTalk is using.
; The routine sets a flag byte, ApTalkPort, and the accumulator to indicate
; which port (if any) AppleTalk is using.
;    $00 = AppleTalk is not using a serial port
;    $01 = AppleTalk is using serial port 1 (printer port)
;    $02 = AppleTalk is using serial port 2 (modem port)
; Note:  This method should be used under ProDOS 8 only.  Under GS/OS, use the
;        .AppleTalk driver's GetPort DStatus subcall.
;
; Enter routine in emulation mode
;
                    longa off
                    longi off
                    mcopy 2/AInclude/M16.MiscTool

WhichPort           start

IDROUTINE           equ $FE1F           returns system ID information

                    stz ApTalkPort      default to not AppleTalk

                    jsr IDROUTINE       call to the system ID routine
                    cpy #$03
                    bcs NewIIGS

OldIIGS             anop                this is a pre-ROM 03 IIGS
                    clc                 to native mode
                    xce
                    rep #$30            16 bit m and x
                    longa on
                    longi on

                    pea $0000           space for result
                    pea $0021           Slot 1 setting
                    _ReadBParam         read battery RAM parameter
;                                         (2 byte result left on stack)

                    pea $0000           space for result
                    pea $0027           Slot 7 setting
                    _ReadBParam         read battery RAM parameter
                    pla                 get slot 7 setting (2 bytes)

                    sec                 emulation mode
                    xce
                    longa off
                    longi off

                    beq FindYourCard    AppleTalk is active
                    pla                 remove slot 1 setting LSB (1 byte)
                    bra OldExit

FindYourCard        inc ApTalkPort      default to port 1
```

```
                    pla                 is slot 1 "your card"? (1 byte)
                    beq ItsPort2        no, must be port 2
                    bra OldExit

ItsPort2            inc ApTalkPort      port 2 is AppleTalk

OldExit             pla                 remove slot 1 setting MSB (1 byte)
                    lda ApTalkPort
                    rts                 return to caller

NewIIGS             anop                ROM 03 or greater IIGS
                    clc                 to native mode
                    xce
                    rep #$30            16 bit m and x
                    longa on
                    longi on

                    pea $0000           space for result
                    pea $000C           port 2 type
                    _ReadBParam         read battery RAM parameter
;                                         (2 byte result left on stack)

                    pea $0000           space for result
                    pea $0000           port 1 type
                    _ReadBParam         read battery RAM parameter
                    pla                 get port 1 setting (2 bytes)

                    sec                 emulation mode
                    xce
                    longa off
                    longi off

                    cmp #$02            is port 1 AppleTalk?
                    bne TryPort2        no
                    inc ApTalkPort      yes
                    pla                 then remove port 2 setting LSB (1 byte)
                    bra NewExit           and exit

TryPort2            pla                 get port 2 setting LSB (1 byte)
                    cmp #$02            is port 2 AppleTalk?
                    bne NewExit         no
                    lda #$02            yes
                    sta ApTalkPort

NewExit             pla                 remove port 2 setting MSB (1 byte)
                    lda ApTalkPort
                    rts                 return to caller

ApTalkPort          entry
                    ds  1               will be 0, 1, or 2
                    end
```

Under GS/OS, use the method shown in the following sample code to determine if AppleTalk is using a serial port:

```
;
; This routine checks to see which serial port, if any, AppleTalk is using.
; The routine sets a flag byte, ApTalkPort, and the accumulator to indicate
; which port (if any) AppleTalk is using.
;    $0000 = AppleTalk is not using a serial port
;    $0001 = AppleTalk is using serial port 1 (printer port)
;    $0002 = AppleTalk is using serial port 2 (modem port)
; Note:  This method should be used under GS/OS only.
;
```

```
; Enter routine in native 16 bit mode
;
                    longa on
                    longi on
                    mcopy 2/AInclude/M16.GSOS
```

```
CheckPort           Start

GetPort             equ $8001           The .AppleTalk DStatus subcall to get
;                                       the port number AppleTalk is currently
;                                       using.

                    phb                 save data bank
                    phk                 data bank = code bank
                    plb

                    lda #$0001          start with device #1
                    sta DIdevNum

FindATDriver        anop
                    _DInfoGS DInfoParms ;call Dinfo
                    bcs DIError         stop searching if error
                    lda DIdeviceIDNum
                    cmp #$001D          is it the AppleTalk main driver?
                    beq ATDriverFound   yes
                    inc DIdevNum        check the
                    bra FindATDriver    next device number

ATDriverFound       anop
                    lda DIdevNum        store device number
                    sta DSdevNum        in the DStatus parm list
                    _DStatusGS DStatusParms ;call DStatus
                    lda portNum         get the port number
                    sta ApTalkPort
                    bra Exit

DIError             anop
;                   cmp #$0011          invalid device number, so the
;                   beq NotFound        AppleTalk main driver wasn't found
;
; Add your code to handle any other errors from DInfo here, because the
; end of the device list was not found.

NotFound            stz ApTalkPort      neither port is in use
                    bra Exit

Exit                anop
                    lda ApTalkPort
                    plb                 restore data bank
                    rtl                 return to caller

ApTalkPort          entry
                    ds 2                will be 0, 1, or 2

DInfoParms          anop
                    dc i2'8'            pCount = 8 parameters
DIdevNum            dc i2'1'            devNum
                    dc a4'NameBuffer'   devName
                    ds 2                characteristics
                    ds 4                totalBlocks
                    ds 2                slotNum
                    ds 2                unitNum
                    ds 2                version
DIdeviceIDNum       ds 2                deviceIDNum

NameBuffer          anop
                    dc i2'31'           Class 1 input string. Max Length=31
                    ds 33
```

```
DStatusParms        anop
                    dc i2'5'            pCount = 5 parameters
DSdevNum            ds 2                devNum
                    dc i2'GetPort'      statusCode = GetPort
                    dc a4'GetPortSList' statusList = GetPortSList
                    dc i4'2'            requestCount = 2
                    ds 4                transferCount

GetPortSList        anop                the GetPort subcall's statusList
portNum             ds 2                $0001 = AppleTalk is using port 1 (printer
port)
;                                       $0002 = AppleTalk is using port 2 (modem port)
                    dc i2'0'

                    end
```

Rule #2: Don't use the SCC Interrupt Handler Vector.

Contrary to what you may have read in a previous version of this Note, you cannot reliably attach your SCC interrupt handler to the SCC Interrupt Handler Vector (vector reference number $0009). The Apple IIGS serial firmware owns the SCC Interrupt Handler Vector (or at least **it** thinks it does). Anytime the serial firmware is used, there is a chance that the serial firmware can grab the SCC Interrupt Handler Vector for its use. CDAs and NDAs that print, the Print Manager tool set, the Text tool set, and the generated GS/OS character drivers associated with the serial ports are examples of code that can and do use the serial firmware.

The only safe place to connect into the interrupt chain is through the operating system. The ProDOS 8 and GS/OS ProDOS 16 call, `ALLOC_INTERRUPT` is the correct place to attach your interrupt handler. The GS/OS `BindInt` call cannot be used to attach your interrupt handler to the SCC Interrupt Handler Vector (VRN $0009) for the same reason that you cannot use the SCC Interrupt Handler Vector directly.

Rule #3: Be very, very careful with SCC Write Register 9 (WR9).

The Z8530 SCC has four registers which are shared by both channels (ports). Of those four, only two are commonly used in the Apple IIGS, `RR3` and `WR9`. `RR3`, which only exists in channel A, lets you check the interrupt pending bits for both SCC channels. `WR9` is the Master Interrupt Control register for both SCC channels and contains the Reset command bits.

You must never reset the channel AppleTalk is using (resetting the channel AppleTalk is using kills AppleTalk). This means you should **never** perform a Force Hardware Reset command (11xxxxxx to `WR9`) even though the *Z8530 Serial Communications Controller Technical Manual* tells you to in the SCC initialization procedure. A hardware reset is performed at system startup, so you shouldn't need to perform a channel reset, even to the channel you are using.

The interrupt control bits (bits D5 - D0) in `WR9` should not be modified (an exception is when you are installing your own SCC interrupt handler). AppleTalk expects the interrupt control bits to always be 001010. If you find the need to perform a channel reset on your channel, remember that the interrupt control bits are programmed at the same time as a channel reset.

Hints for the Serial Adventure

Next are a few hints for those who would like to explore the world of knocking on the registers of the Z8530 SCC.

Hint #1: Synchronize your code with the SCC logic.

Before writing to the SCC chip for the first time, you should make an attempt to ensure your code is synchronized with the SCC's logic. This needs to be done only once when you are initializing the SCC. This can be accomplished with a single read of SCC Read Register 0 (`RR0`). For example, if you're using serial port 2 (the modem port), the following code reads `RR0` of SCC channel B:

```
                    longa off           must be using 8-bit accumulator
                    lda $C038           read RR0 of SCC Channel B
```

Hint #2: Watch out for interrupts from the other SCC channel.

Except for `RR0`, `WR0`, and the two SCC data registers, all SCC registers are accessed in a two-step process. First, the register number you want to select is written to `WR0`. After the register number is set, the next read from or write to the command register accesses the register selected in the first step. Because several of the SCC registers are shared between the two SCC channels and because code accessing them may not always be yours (i.e., AppleTalk), interrupts should be disabled during the two steps. The following code shows two quick subroutines to access the SCC's Read and Write registers while preventing interrupts between the register number set and the register read or write steps:

```
                    longa off           must be using 8-bit accumulator
                    longi off             and index registers
;
; Write to a SCC command register - channel A or B.
; Input:  A = value to store
;         X = SCC register number ($0-$F)
;         Y = $01  channel A
;             $00  channel B
;
WriteSCC            php                 save the current interrupt status
                    sei                 disable interrupts
                    pha                 save value to write
                    txa                 get SCC register number from X
                    sta $C038,y         set the register number
                    pla                 restore value to write
                    sta $C038,y         write the value
                    plp                 restore the interrupt status
                    rts
```

```
;
; Read from a SCC command register - channel A or B.
; Input:  A = SCC register number ($0-$F)
;         Y = $01  channel A
;             $00  channel B
; Output: A = register value
;
ReadSCC             php                 save the current interrupt status
                    sei                 disable interrupts
                    sta $C038,y         set the SCC register number
                    lda $C038,y         get the value from the SCC register
                    xba                 look ahead 2 lines...
                    plp                 restore the interrupt status
                    xba                 set N and Z flags for exit
                    rts
```

Just to be complete, here's how `RR0`, `WR0`, the receive buffer, and the transmit buffer SCC registers are accessed on the Apple IIGS:

```
                    longa off           must be using 8-bit accumulator
                    longi off             and index registers
;
; Read RR0 - channel A or B
; Input:  Y = $01  channel A
;             $00  channel B
; Output: A = RR0 register value
;
ReadRR0             lda $C038,y         get the value from RR0
                    rts
;
; Write WR0 - channel A or B
; Input:  A = value to store at WR0
;         Y = $01  channel A
;             $00  channel B
;
WriteWR0            sta $C038,y         write the value to WR0
                    rts
;
; Read from SCC receive buffer - channel A or B
; Input:  Y = $01  channel A
;             $00  channel B
; Output: A = value of data received
;
ReadData            lda $C03A,y         get the value from SCC data register
                    rts
;
; Write to SCC transmit buffer - channel A or B
; Input:  A = value of data to transmit
;         Y = $01  channel A
;             $00  channel B
;
WriteData           sta $C03A,y         write the value to SCC data register
                    rts
```

Hint #3: All SCC channels are not created equal.

In the IIGS, the SCC's receive and transmit clocks for both channels are driven by a single crystal oscillator circuit. This is accomplished by connecting a 3.6864 MHz crystal between the /RTxC and /SYNC pins of channel A. Channel B's /RTxC pin is connected to Channel A's /SYNC pin to drive channel B's clocks from channel A's oscillator circuit.

Because of this single circuit, Write Register 11 (WR11) bit 7 must be set to 1 for SCC channel A and must be set to 0 for SCC channel B.

Hint #4: RR3 is available only in SCC channel A.

When your interrupt handler is checking to see if the interrupt condition was caused by your SCC channel, remember to always look at RR3 in SCC channel A. RR3 in channel A contains the interrupt pending bits for both SCC channels. RR3 in channel B always returns all zeros, which doesn't tell you a lot about what's happening.

Don't be a Serial Killer

How to Install and Remove your SCC Interrupt Handler

If you're going to handle serial I/O and don't want your application to have to poll the SCC chip all the time to see if something has happened, you probably want to install an interrupt handling routine that is called every time a SCC chip condition you want to know about occurs. This section of the Note shows how to install and remove your own SCC interrupt handler.

The steps for installing your SCC interrupt handler are:

1. Ensure the serial firmware's Input and Output buffering is disabled. The state of I/O buffering can be checked by looking at bit 14 of the ModeBitImage parameter returned by the GetModeBits extended interface call. I/O buffering can be disabled with the firmware's BD control command.
2. Disable the SCC Master Interrupt Enable (WR9, bit 3) briefly while performing the next six steps. The value you should write to WR9 is 00000010.
3. Get the address of the system interrupt flag byte, SerFlag. The ROM version determines the method of finding the address of SerFlag. In ROM version 01 and later, you can get the address with a call to the Miscellaneous Tools GetAddr using a reference number of $000E. With ROM version 00 (the original IIGS ROM), the address of SerFlag is $E10104. Refer to the Apple II Miscellaneous Technical Note #7, Apple II Family Identification for information on identifying Apple IIGS ROM versions.
4. Once you have the correct address of SerFlag, preserve the byte's current value, then turn on the bits in the byte which reflect the port from which you are handling interrupts. The bits for the different ports are as follows (note the relationship of the bits of RR3 to SerFlag):

```
Port 1:    ORA    #%00111000
Port 2:    ORA    #%00000111
```

5. Initialize the SCC modes. The *Z8530 Serial Communications Controller Technical Manual* shows the order the SCC registers must be programmed. However, you must stray from the manual slightly due to the hardware implementation of the SCC in the IIGS. A typical initialization sequence to set the SCC up for asynchronous serial communications through channel B (the modem port) would look similar to the following:

SCC Register	Value	Comment
RR0	-	ensure synchronization with SCC
WR4	01000100	x16 clock, 1 stop, no parity
WR3	11000000	8 bit receive data, auto enables off, receiver disabled
WR5	01100010	DTR is active, 8 bit transmit data, no break, transmit disabled, RTS is inactive
WR11	01010000	no Xtal on channel B, receive and transmit clock = baud rate generator output
WR12	01011110	low byte of baud rate generator time constant = $5E - 1200 baud
WR13	00000000	high byte of baud rate generator time constant = $00 - 1200 baud
WR14	00000000	no local loopback or auto echo, /DTR follows inverted DTR bit in WR5, use /RTxC for baud rate generator clock, disable baud rate generator
WR14	00000001	enable the baud rate generator
WR3	11000001	receiver enabled
WR5	01101010	transmit enabled
WR15	00000000	no interrupts on this channel for now...

6. Tell the SCC which external and status conditions can cause an interrupt by setting the appropriate bits in WR15. This step is not needed unless you are setting bit 0 of WR1 (External/Status Master Interrupt Enable) in the next step.
7. Enable the interrupts modes you want by setting the appropriate bits in WR1 (00010011 for all SCC interrupt conditions).
8. Use ALLOC_INTERRUPT to add your interrupt handler to the operating system's interrupt vector table. The interrupt identification number returned by ALLOC_INTERRUPT is needed when you remove your interrupt handler.
9. Reenable the SCC Master Interrupt flag (WR9, bit 3). The value you should write to WR9 is 00001010.

The interrupt handling routine must conform to the rules listed in the *ProDOS 8 Technical Reference Manual* and *GS/OS Reference*, Volume 2.

When you get ready to shut down your application, you need to remove your interrupt handler. The steps for removing the SCC interrupt handler you installed are as follows:

1. Disable the SCC Master Interrupt Enable (`WR9`, bit 3) briefly while performing the next six steps. The value you should write to `WR9` is 00000010.
2. Disable all interrupts modes for your port by writing a $00 to `WR1`.
3. Remove any character that might be left in the receive data register by reading it once.
4. Clear any pending transmit overrun and external and status interrupts by writing 11010000 to `WR0`.
5. Clear any pending transmit interrupt by writing 00101000 to `WR0`.
6. Use `DEALLOC_INTERRUPT` to remove your interrupt handler from the operating system's interrupt vector table.
7. Restore `SerFlag` to its original value.
8. Reenable the SCC Master Interrupt flag (`WR9`, bit 3). The value you should write to `WR9` is 00001010.

Further Reference

- *Apple IIGS Toolbox Reference Manual*, Volume 1
- *Apple IIGS Firmware Reference Manual*
- *Apple IIGS Hardware Reference Manual, Second Edition*
- *GS/OS Reference*, Volumes 1 and 2
- *ProDOS 8 Technical Reference Manual*
- Apple II Miscellaneous Technical Note #7, Apple II Family Identification
- GS/OS Technical Note #9, Interrupt Handling Anomalies
- *Z8530 Serial Communications Controller Technical Manual* (Zilog Corporation)
- *Z85C30 Serial Communications Controller Technical Manual* (Advanced Micro Devices, Inc.)